Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 507426, 6 pages
doi:10.1155/2009/507426

Research Article

An FPGA Implementation of a Parallelized MT19937 Uniform Random Number Generator

Vinay Sriram and David Kearney

University of South Australia, Reconfigurable computing Laboratory, School of Computer and Information Science, Mawson Lakes Campus, Adelaide, SA 5085, Australia

Correspondence should be addressed to Vinay Sriram, srivb001@students.unisa.edu.au

Received 20 August 2008; Revised 16 February 2009; Accepted 21 April 2009

Recommended by Miriam Leeser

Recent times have witnessed an increase in use of high-performance reconfigurable computing for accelerating large-scale simulations. A characteristic of such simulations, like infrared (IR) scene simulation, is the use of large quantities of uncorrelated random numbers. It is therefore of interest to have a fast uniform random number generator implemented in reconfigurable hardware. While there have been previous attempts to accelerate the MT19937 pseudouniform random number generator using FPGAs we believe that we can substantially improve the previous implementations to develop a higher throughput and more area-time efficient design. Due to the potential for parallel implementation of random numbers generators, designs that have both a small area footprint and high throughput are to be preferred to ones that have the high throughput but with significant extra area requirements. In this paper, we first present a single port design and then present an enhanced 624 port hardware implementation of the MT19937 algorithm. The 624 port hardware implementation when implemented on a Xilinx XC2VP70-6 FPGA chip has a throughput of 119.6×10^9 32 bit random numbers per second which is more than 17x that of the previously best published uniform random number generator. Furthermore it has the lowest area time metric of all the currently published FPGA-based pseudouniform random number generators.

1. Introduction

Reconfigurable computing is increasingly being seen as an attractive solution for accelerating simulations that require fast generation of large quantities of random numbers. Although random numbers are often a very small part of these algorithms inability to generate them fast enough, them can cause a bottleneck in the reconfigurable computing implementation of the simulation. For example, in the simulated generation of infrared scenes that take into account the effects of a turbulent atmosphere and the effects of CCD camera sensor electronic noise, each 352×352 scene generated by the simulation requires more than 1.87×10^6 gaussian random numbers. A real-time simulation sequence at 15 scenes/s thus needs more than 28.1×10^6 random samples generated per second. Since a typical software uniform generator [1] can only manage 10×10^6 per second you would need 29 PCs to keep up with this rate.

A key requirement of Infrared (IR) scene simulation is the necessity to generate large sequences of random numbers on the provision of a single seed. Not all random number generators are capable of doing this (e.g., see those presented in [2]). Moreover, in order to achieve the high throughput required, it is important to make use of algorithm's internal parallelism (i.e., by splitting the algorithm into independent subsequences) as well as external parallelism (i.e., through parallel implementations of the algorithm). It has been recommended in [3] and reinforced in [4] that in order to prevent possible correlations between output sequences in parallel implementation of the same algorithm using different initial seeds, it is necessary to use a random number generator that has a period greater than 2^{200}. In summary the requirements of an FPGA optimized uniform random number generator for IR scene simulation are as follows:

(1) should be seeded random number generator (so that the same sequence may be regenerated);

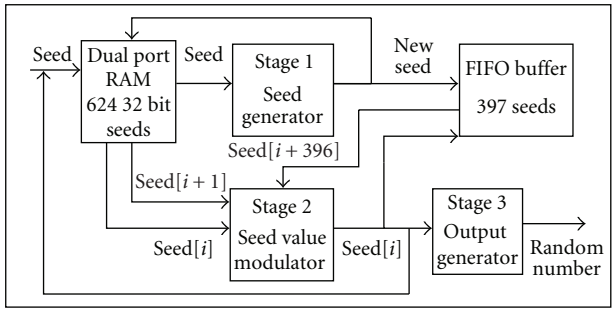


Figure 1: Single port version.

(2) have the ability to generate a large quantity of random numbers from one seed;

(3) can be split in many independent subsequences;

(4) have a very large period;

(5) generate random numbers quickly;

(6) satisfy statistical tests from randomness;

(7) able to generate parallel streams of uncorrelated random numbers.

2. Survey of FPGA-Based Uniform Random Number Generators

As discussed in the previous section IR scene simulation [5] requires fast generation of large sequences of random numbers on the provision of a single seed. From the extensive literature in the field of software pseudouniform random number generators, some algorithms that achieve this are the generalized feedback shift register and the MT19937. They both have the ability to generate large sequences of random numbers on the provision of a single seed and have the ability to be split in independent subsequences to allow for a more parallelized implementation in hardware. An additional benefit of these algorithms is that they have large periods. It has been recommended in [1] and reinforced in [4] that in order to prevent possible problems with correlations when implementing the same algorithm with different initial seeds in parallel, the algorithm needs to have a period in excess of 2^{200}. The MT19937 algorithm has a period of 2^{19937}, which therefore allows for parallel implementation of MT19937 algorithm with different initial seeds.

There are currently two FPGA optimized implementations of the MT19937, including a single port design, see [6], and a multiport design presented in [7]. The well-known generalized feedback shift register has been modified for FPGA implementation in [8] to achieve the smallest area time design to date. Thus it is of interest to see if a hardware implementation of a 624 port MT19937 algorithm can be made more competitive. This is the subject of investigation of this paper. This paper is organized as follows, in Section 2 the MT19937 algorithm is briefly described. In Sections 3 and 4 we present single port and 624 port hardware implementations of the MT19937 algorithm. In Section 5, diehard test results of the two hardware implementations along with the performance comparisons of these implementations with other published FPGA-based pseudouniform random number generators are presented.

3. MT19937

The origins of the MT19937 algorithm are from the Tausworthe generator, which is a linear feedback shift register that produces long sequences of binary bits; see [9]. The period of this polynomial, which is irreducible, depends on the characteristic polynomial. The period is the smallest integer n for which x^n+1 is divisible by the characteristic polynomial. The polynomial has the following form;

$$x_{n+1} = \left(A_1 x_n + A_2 x_{n-1} + \cdots + A_k x_{n-k+1}\right)^{\bmod 2}, \quad (1)$$

where x_i, $A_i \in [0,1]$ for all i. Although this algorithm produces uniform random bits, it is slow. This algorithm was later modified by Lewis and Payne in [10], by creating a variant of this known as the generalized feedback shift register.

$$x_i = (x_i - p)x \quad \text{or} \quad (x_i - q), \quad (2)$$

where each x_i is a vector of size w with components 0 or 1. The maximum possible period of $2p-1$ of this generator is achieved when the primitive trinomial $xp+xq+1$ divides $xn-1$ for $n = 2p-1$, for the smallest value of n. The maximum period can be achieved by selecting n as a Mersenne Prime. It was later identified that the effectiveness, that is, the randomness of the numbers produced, of this algorithm was dependent on the selection of initial seeds. Furthermore, the algorithm required n words working area (which was memory consuming) and the randomness of the numbers produced was dependent on the selection of initial seeds. This discovery led Matsumoto and Kurita 1994 to further revise this algorithm to develop the twisted generalized feedback shift register II in [11]. This generator used linear combinations of only relatively few bits of the preceding numbers and was thus considerably faster and was named TT800. Later Matsumoto and Kurita in 1998 further revised the TT800 to admit a Mersenne-prime period, and this new algorithm was called the MT19937; see [12].

The MT19937 algorithm generates sequences of uniformly distributed pseudo random integers 32 or 54 bit numbers between $[0, 2w-1)$. The MT19937 algorithm is based on the following linear recurrence formula, where x and a denote word vectors, and A is w by w matrix. The proof of the algorithm is provided in [12],

$$x_{k+n} = x_{k+m} \otimes \left(x_k^u \mid x_{k+1}^l\right), \quad (3)$$

where $k = (0, 1, \ldots)$.

4. Single Port Version

This section describes our first hardware implementation of MT19937 which we call the single port version. Generation

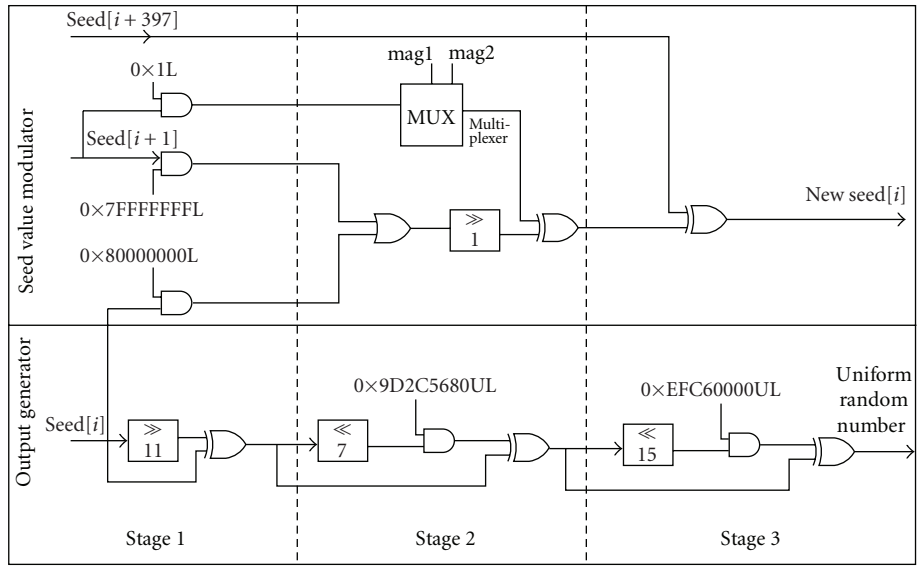


Figure 2: Internal logic of stage 2 and stage 3.

of random numbers is carried out in 3 stages, namely, the seed generator, seed value modulator, and output generator. This is illustrated in Figure 1.

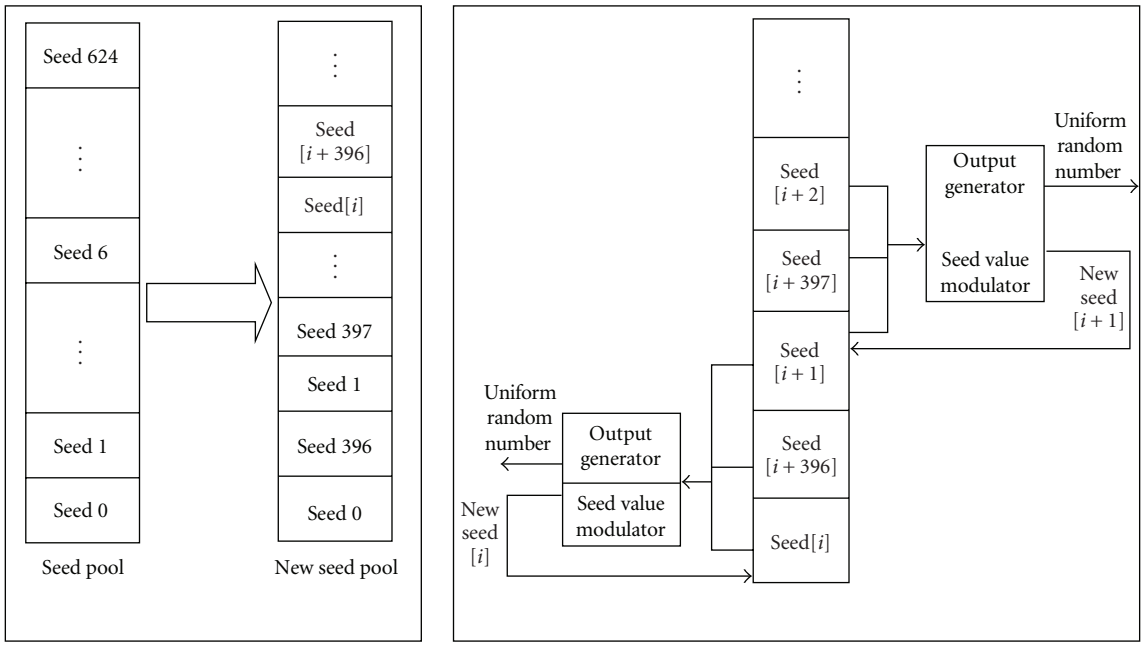
Typically the user provides one number as a seed; however, the MT19937 algorithm works with a pool of 624 seeds so that generator stage generates 624 seeds from the single input from the user. In stage two (the seed value modulator), which is the core of the algorithm, three values seed[i], seed[$i + 1$], and seed[$i + 396$] are read from the pool and based on the computation defined in the algorithm; seed[i] is updated. In the final stage, the output generator reads one of the pool values and generates the output uniform random number from this value.

The logic used to generate values out of stages 2 and 3 is shown in Figure 2. The simplest form of parallelism for MT19937 is to perform stages 2 and 3 in parallel, and this is illustrated in Figure 2. Note that it is not possible to more finely pipeline the output generator because its processing rate is tied to the seed value modulator, which can only be pipelined into 3 stages. In other words, the seed value modulator is a bottleneck in the design. It needs to be pointed out that if the data comes from a dual port BRAM only one value can be read and one written in the same clock cycle. Since we need three values to be read, we use 3 dual port BRAMs. We then need logic to decide which BRAM to write into. The write back selection logic forms another stage in the seed value modulator, which now has 4 stages. Not shown in Figure 1 is the logic by which the BRAM address will be read from and written to. The single port version generates one new random number per clock cycle. In Figure 2, mag1, mag2, and the hex numbers are constants given in the algorithm definition.

The single port version provided is similar to the software implementation of the MT19937 algorithm as it does not provide any significant parallelization in the generation of the seeds. The only parallelism that is exploited is in the concurrent execution of seed value modulator (stage 2) and output generator (stage 3). It was also found that it was not possible to pipeline the output generator to more than 3 stages as it was tied to the seed value modulator. Significant improvements in throughput could be achieved by the parallelization of the stages 2 and 3 in addition to executing them in parallel as shown above. However, the

Table 1: Diehard test results.

Test	Single-port implementation	624 port implementation
Birthday	0.348414	0.467321
OPERM5	0.634231	0.892018
Binary Rank (31×31)	0.523537	0.678026
Binary Rank (32×32)	0.521654	0.578317
Binary Rank (6×8)	0.235435	0.457192
Bitstream	0.235891	0.280648
OPSO	0.624894	0.987569
OQSO	0.235526	0.678913
DNA	0.623498	0.446857
Stream Count-the-1	0.352235	0.789671
Byte Count-the-1	0.652971	0.865671
Parking Lot	0.662841	0.567193
Minimum Distance	0.623121	0.467819
3D Spheres	0.622152	0.678991
Squeeze	0.951623	0.456719
Overlapping Sums	0.542882	0.345671
Runs Up	0.626844	0.456191
Runs Down	0.954532	0.898761
Craps	0.347221	0.689187



(a) Storage of seeds pools

(b) An example of one of the parallel instances of the 624 port design

Figure 3: 624 port version.

problem with parallelizing stages 2 and 3 is that currently the seeds are all stored in a single dual port BRAM. It is not possible to carry out multiple reads and multiple writes to a single BRAM in one clock cycle. Previously in [7] parallelization of both these stages was achieved by dividing the seeds into multiple BRAMs. This however significantly increased the area requirements of the design. In the next section we study this problem in more detail and present our new design that has a high throughput and is area efficient.

5. 624 Port Version

There has previously been an attempt to parallelize the MT19937 algorithm by dividing the seeds into various pools and then replicating the stages 2 and 3 to generate multiple outputs; see [7]. However, it was noted that this was found not to be area time efficient. A close examination of the design reveals that in order to parallelize the generation of random numbers, the authors divide the seeds into multiple BRAMs. Although this did increase the throughput, it greatly increased the area of the design as well. The reason for this is that the logic required to generate the necessary BRAM address increased in complexity with the dividing of seeds across multiple BRAMs.

It is important to note here that the problem is not the parallelization of the generation of the uniform random numbers but is the storing of seeds in multiple BRAMs. Thus if the seeds were to be stored in registers rather than BRAMs the logic used to generate the BRAM address could be saved. The excessive use of BRAMs to store seeds was always considered problematic. For example, in [13] it was found that the TT800 algorithm suffered in a similar manner when the seeds were distributed across multiple BRAMs. In this paper it was reported that the single port version used 81 Xilinx slices while the 3 port one used 132 slices. Of the 132 slices used, 60 slices were used for the complex BRAM address generation logic. We believe that we can parallelize the MT19937 algorithm to the maximum possible 624 port by storing seeds in registers rather than BRAMs. In this section, we study a more simplified design for a 624 port MT19937 random number generator that uses registers to store seeds.

A careful examination of the addressing scheme shows that the seeds can be divided into groups in such a way that there is no need for the logic in one group to access the seeds in another group. We call these groups seed pools and these are shown in Figure 3.

We also present a generic model which makes use of each of these seed pools to modify the seed value and generate new random numbers per clock cycle. Now on each seed pool the two stages of the MT19937 presented in Figure 3 work together to modify each seed value and generate a new one. This is illustrated in Figure 3(b). From a point of view of circuit speed and complexity, no register is shared by more than two reading channels and one writing channel. The consequence is that register access logic is simpler, smaller, and faster.

6. Results

6.1. Test for Randomness. As a preliminary test, the output of the hardware implementations was successfully verified against the output of the software implementation. For a more complete test, the hardware implementations have

Table 2: Period, area, time, and throughput comparisons.

	Period	Xilinx XC2VP70 Slices	LUTs	Clock rate MHz	Area time slices × sec per 32 bit number $\times 10^{-6}$	Throughput 32 bit numbers/sec $\times 10^9$
MT19937 [This work]						
Single port	2^{19937}	87	—	319	0.34	0.24
624 port	2^{19937}	1253	—	190	0.009	119.6
Software* [12]	2^{19937}	—	—	2800	—	1.017
MT19937 [6]	2^{19937}	420	—	76	5.5	0.076
MT19937 [7]						
SMT‡	2^{19937}	149	—	128.02	1.16	0.12
PMT52‡	2^{19937}	2.850	—	71.63	0.76	3.7
FMT52‡	2^{19937}	11.463	—	157.60	1.45	8.2
$PMT52_in$‡	2^{19937}	2.914	—	62.24	0.9	3.2
$FMT52_in$‡	2^{19937}	5.925	—	74.16	0.77	3.8
LUT [8]						
4-tap, $k = 32$	2^{32}	—	33	309	0.06†	0.3
4-tap, $k = 64$	2^{64}	—	65	310	0.05†	0.6
4-tap, $k = 96$	2^{98}	—	97	298	0.05†	1.1
4-tap, $k = 128$	2^{128}	—	127	287	0.05†	1.8
4-tap, $k = 256$	2^{258}	—	257	246	0.06†	1.8
4-tap, $k = 1248$	2^{1248}	—	1249	168	0.09†	6.7
3-tap, $k = 32$	2^{32}	—	33	302	0.06†	0.31
3-tap, $k = 64$	2^{64}	—	65	319	0.05†	0.64
3-tap, $k = 96$	2^{98}	—	97	308	0.06†	1.2
3-tap, $k = 128$	2^{128}	—	127	287	0.06†	1.7
3-tap, $k = 256$	2^{258}	—	257	243	0.07†	1.9
3-tap, $k = 1248$	2^{1248}	—	1249	173	0.09†	6.7

* Software implementation was on a Pentium 4 (2.8 GHz) single core processor.
† Each slice consists of 2 LUTs, therefore the area time rating of these desings equals LUTs/2 ∗ Time.
‡ This design has been implemented on an Altera Stratix. Each Xilinx slice is equivalent to two Altera logic elements.

been tested using the diehard test. In Table 1 the test results of the diehard tests are presented. The diehard test produces P-values in the range $[0, 1)$. The P-values are to be above .025 and below .975 for the test to pass. Both the implementations pass this test.

6.2. Comparison with Existing FPGA-Based Uniform Random Number Generators. In this section we compare our designs with those that are currently published. We compare our designs on the basis of area time rating and throughput. In contrasting these solutions we take into account the amount of total resources used, including slices, LUTs, and flip flops.

From Table 2 it should be noted that our 624 port hardware implementation of the MT19937 algorithm when implemented on a Xilinx XC2VP70-6 FPGA chip achives more than 115x the throughput of the same algorithm's implementation in software on a Pentium 4 (2.8 GHz) single core processor. It can also be seen that there are no other published random number generators from current literature that are able to achieve a throughput of greater than 119×10^9 32 bit random numbers per second. The closest competitors are the FMT52, 4-tap, $k = 1248$, and 3-tap, $k = 1248$ random number generators which are still significantly behind. The design presented herein has an AT rating of only 0.009×10^{-9} for a throughput of 119×10^9 random numbers per second. A further criticism of [8] is that the specialized feedback register matrix used in the implementation was not completely published.

Our best implementation, which is the 624 port MT19937, uses only 1253 Xilinx slices. This is significantly less than all of the other multiport designs currently published in literature as we use registers to store seeds and have arranged our seed value modulator and output generator pipelines in an area efficient manner. Thus we do not require any complex BRAM address generation logic and access to BRAMs. As a result we save on area and since our design if 624 port we generate 624 uniform random numbers per clock cycle. In a reconfigurable computing implementation, where only the random number generation is accelerated in hardware, like all of the other FPGA-based random number generators, the 624 port implementation is limited by the I/O bandwidth of the FPGA.

7. Conclusion

In this paper we have presented a unique 624 port MT19937 hardware implementation. Whilst currently there are hardware implementations of uniform random number generators published none seem to be able to offer a high throughput as well as area time efficiency. It was demonstrated that the 624 port design presented in this paper is a high throughput, area time efficient FPGA optimized pseudouniform random number generator with a large period and with the ability to generate large quantities of uniform random numbers from a single seed. Thus making suitable for use in a reconfigurable computing implementation of real-time IR scene simulation.

Acknowledgment

Research undertaken for this report has been assisted with an international scholarship from the Maurice de Rohan fund. This support is acknowledged and greatly appreciated.

References

[1] P. L'Ecuyer, "Random number generation," in *Handbook of Simulation*, J. Banks, Ed., chapter 4, pp. 93–137, John Wiley & Sons, New York, NY, USA, 1998.

[2] W. H. Press, B. P. Flannery, et al., *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK, 1986.

[3] A. Srinivasan, M. Mascagni, and D. Ceperley, "Testing parallel random number generators," *Parallel Computing*, vol. 29, no. 1, pp. 69–94, 2003.

[4] P. L'Ecuyer and R. Panneton, "Fast random number generators based on linear recurrences modulo 2: overview and comparison," in *Proceedings of the Winter Simulation Conference*, pp. 110–119, IEEE Press, 2005.

[5] V. Sriram and D. Kearney, "High speed high fidelity infrared scene simulation using reconfigurable computing," in *Proceedings of the IEEE International Conference on Field Programmable Logic and Applications*, IEEE Press, Madrid, Spain, August 2006.

[6] V. Sriram and D. A. Kearney, "An area time efficient field programmable mersenne twister uniform random number generator," in *Proceedings of the International Conference on Engineering of Reconfigurabe Systems and Algorithms*, CSREA Press, June 2006.

[7] S. Konuma and S. Ichikawa, "Design and evaluation of hardware pseudo-random number generator MT19937," *IEICE Transactions on Information and Systems*, vol. E88-D, no. 12, pp. 2876–2879, 2005.

[8] D. B. Thomas and W. Luk, "High quality uniform random number generation through LUT optimised linear recurrences," in *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT '05)*, pp. 61–68, Singapore, December 2005.

[9] R. Tausworthe, "Random numbers generated by linear recurrence modulo two," *Mathematics of Computation*, vol. 19, pp. 201–209, 1965.

[10] T. Lewis and W. Payne, "Generalized feedback shift register pseudorandom number algorithm," *Journal of the ACM*, vol. 20, no. 3, pp. 456–468, 1973.

[11] M. Matsumoto and Y. Kurita, "Twisted GFSR generators–II," *ACM Transactions on Modeling and Computer Simulation*, vol. 4, no. 3, pp. 254–266, 1994.

[12] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.

[13] V. Sriram and D. Kearney, "Towards a multi-FPGA infrared simulator," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 4, no. 4, pp. 50–63, 2007.